***23. Характеристики алгоритма Sapphire***

Jones, Hosking, Moss — The Garbage collection handbook

17.6 Sapphire стр. 345

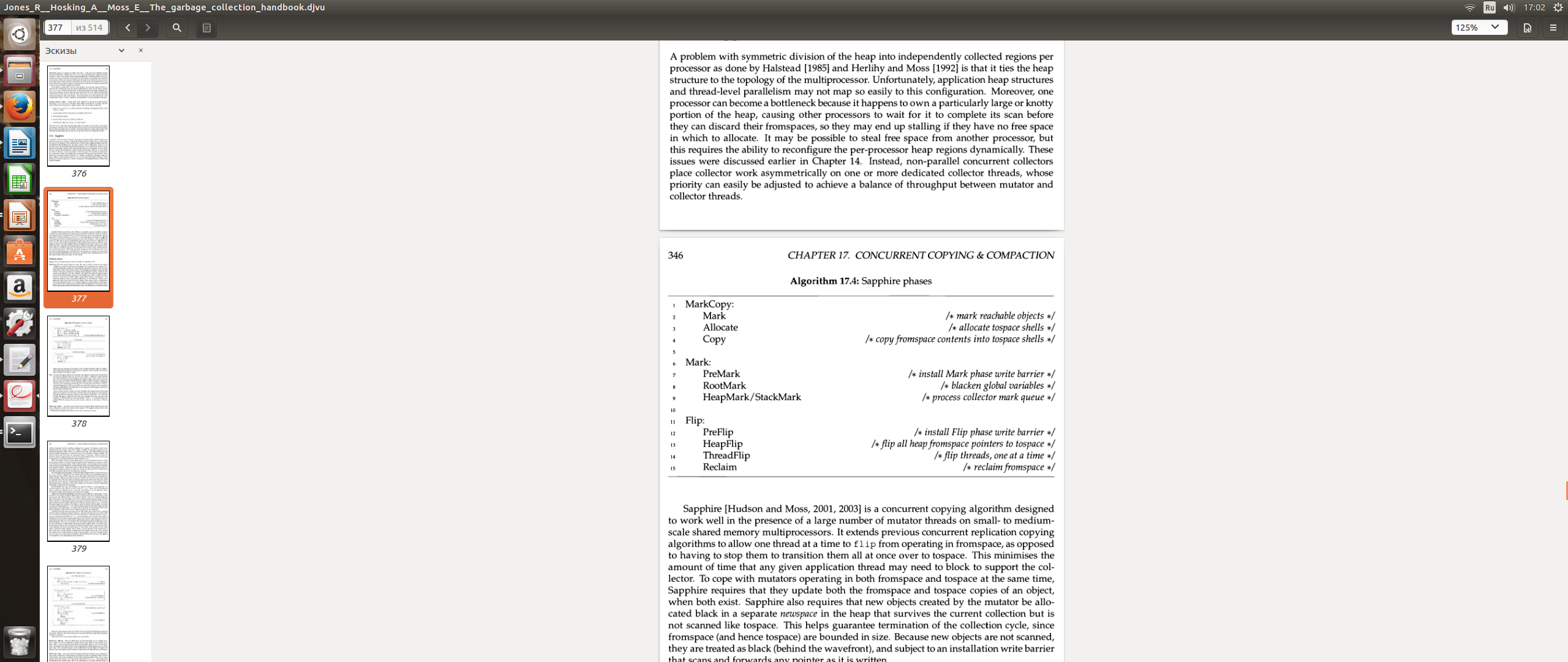
[норм ссылка](ftp://ftp.cs.umass.edu/pub/osl/papers/sapphire-2003.pdf) – тут понятней и есть термины с пояснением

Эквивалентность объектов в “From” и “To”

* Объекты в целевом и исходном пространстве гарантированно имеют одинаковые значения полей
* Реализуется при помощи Write барьера обновляющего значения полей в обоих пространствах при наличии ссылки на новое положение в заголовке объекта

Не-параллельный конкурентный сборщик выполняет работу ассиметрично на одной и более выделенных нитей сборщика, чья очередность может быть легко отрегулирована, дабы достичь баланса между нитями мутатора и сборщика.

Фазы алгоритма:

Sapphire — конкурентный копирующий алгоритм, разработанный чтобы эффективно работать в присутствии большого количества нитей мутатора на small- medium-scale мультипроцессорах с разделяемой памятью. Он позволяет одной нити за раз сделать кувырок(flip), перенеся деятельность из пространства fromspace, в противовес тому, чтобы остановить их для перенесения их всех в пространство tospace. Это минимизирует количество времени, на которое любая нить данного приложения может быть заблокирована, дабы сборщик отработал. Чтобы совладать с мутаторами, действующими одновременно и в tospace и с fromspace одновременно, требуется, чтобы они обновляли обе копии объекта, когда обе существуют. Sapphire также требует, чтобы новые объекты, созданные мутатором размещались бы черными(см. трехцветная раскраска) в отдельном newspace в куче, которое переживает текущую сборку, но не сканируется как tospace. Это помогает гарантировать завершение цикла сборки, поскольку fromspace (и tospace) ограничены в размерах. Поскольку новые объекты не сканируются, к ним относятся как к черным, и ставят write барьер который сканирует и пересылает любой указатель, если он записывается.

Выше перевод вот этого:

Sapphire [Hudson and Moss, 2001, 2003] is a concurrent copying algorithm designed to work well in the presence of a large number of mutator threads on small- to medium- scale shared memory multiprocessors. It extends previous concurrent replication copying algorithms to allow one thread at a time to flip from operating in fromspace, as opposed to having to stop them to transition them all at once over to tospace. This minimises the amount of time that any given application thread may need to block to support the collector. To cope with mutators operating in both fromspace and tospace at the same time, Sapphire requires that they update both the fromspace and tospace copies of an object, when both exist. Sapphire also requires that new objects created by the mutator be allocated black in a separate newspace in the heap that survives the current collection but is not scanned like tospace. This helps guarantee termination of the collection cycle, since fromspace (and hence tospace) are bounded in size. Because new objects are not scanned, they are treated as black (behind the wavefront), and subject to an installation write barrier that scans and forwards any pointer as it is written.